University of Antwerp
Faculty of Business and Economics

DEPARTMENT OF ENGINEERING MANAGEMENT

The real-time on-demand bus routing problem: what is the cost of dynamic requests?

Lissa Melis & Kenneth Sörensen

UNIVERSITY OF ANTWERP
Faculty of Business and Economics
City Campus
Prinsstraat 13, B.226
B-2000 Antwerp
Tel. +32 (0)3 265 40 32
www.uantwerpen.be

FACULTY OF BUSINESS AND ECONOMICS

DEPARTMENT OF ENGINEERING MANAGEMENT

The real-time on-demand bus routing problem: what is the cost of dynamic requests?

Lissa Melis & Kenneth Sörensen

RESEARCH PAPER 2021-003
AUGUST 2021

University of Antwerp, City Campus, Prinsstraat 13, B-2000 Antwerp, Belgium
Research Administration – room B.226
phone: (32) 3 265 40 32
e-mail: joeri.nys@uantwerpen.be

The research papers from the Faculty of Business and Economics are also available at www.repec.org (Research Papers in Economics - RePEc)

D/2021/1169/003

The real-time on-demand bus routing problem: what is the cost of dynamic requests?

Lissa Melis, Kenneth Sörensen

August 2021

Abstract

The real-time on-demand bus routing problem (ODBRP) supports the online routing of buses in a large-scale ride-sharing system. Given are a set of buses with fixed capacity, a set of bus stations and a set of transportation requests, only part of which are known before the planning horizon. A request consists of a set of possible departure and arrival stations, as well as an earliest departure and latest arrival time. The aim is to (1) assign each passenger to a departure and arrival bus station and (2) develop a set of bus routes to fulfill each request within its time window while minimizing the total user ride time.

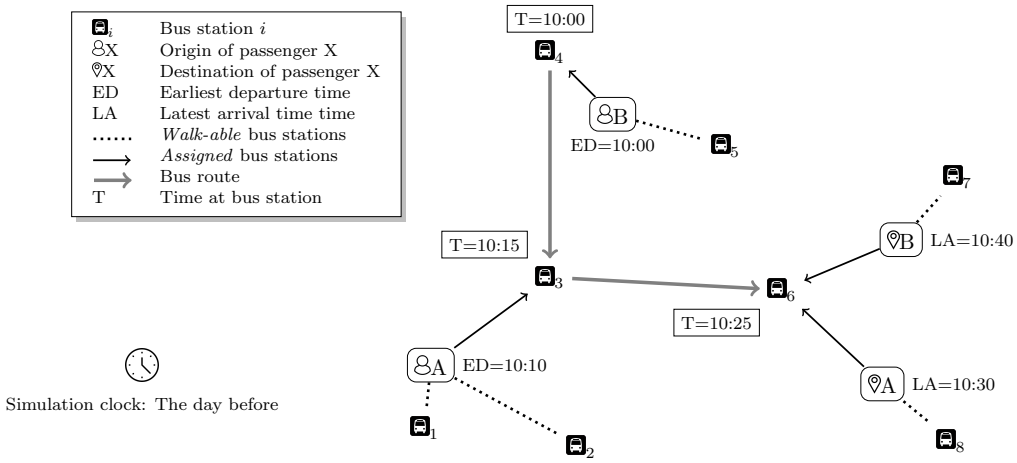
Including the possibility for requests to be issued *after* the start of the planning horizon, i.e., when buses have already started servicing other requests, requires a dynamic re-optimization of a partially executed solution. Compared to the case in which all requests are known beforehand, the solution quality, expressed as the total user ride time, is expected to decline. This decline in objective function value can be seen as the "cost" of the dynamic requests. In this paper, we introduce the real-time ODBRP as a new optimization problem and present a heuristic to deal with dynamic requests. In addition, an extensive set of experiments allows us to conclude that dynamic requests indeed lead to higher user ride times, especially for passengers who submit their request at the last minute. Passengers are therefore encouraged to send their request well in advance, as this results in lower and more stable user ride times, higher customer satisfaction, and higher revenues for the operating on-demand bus company.

Keywords— Public transport, Transportation, Metaheuristic, Mobility on demand

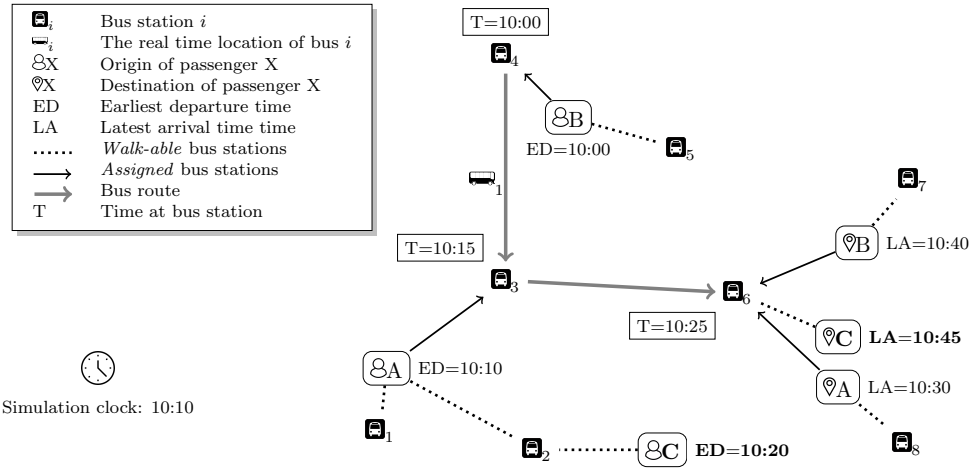
1 Problem statement

Recent developments in automated vehicle location (AVL) systems which provide the location of vehicles in real-time, and the use of smartphones with incorporated geolocation technologies, create a window of opportunity for new mobility services. As a result of these technologies, a shift from traditional car ownership and the use of traditional public transport to services like ride-sharing (e.g., UberPool), car-sharing (e.g., Poppy) and on-demand transport options or ride-hailing (e.g., Uber, Lyft) can be observed. These relatively new services, categorized under the umbrella term *mobility-on-demand (MOD)*, are more flexible compared to traditional public transport lines and they are beneficial to counteract congestion in cities (Tsao et al., 2019; Yan et al., 2019). However, these examples remain limited in scale, while the recent technologies mentioned would, at least in theory, enable a large-scale shift towards on-demand public transport. In such a system, dynamic bus routes would be created completely based on the customers' demand, eliminating the need for fixed routes and timetables. As bus routes and timetables are completely determined based on transportation requests, the efficiency of the system can be optimized both from the perspective of the passenger (e.g., minimal user ride times) and from that of the bus company (e.g., minimal number of kilometres driven empty) (Melis and Sörensen, 2020; Navidi et al., 2018; Bischoff et al., 2018).

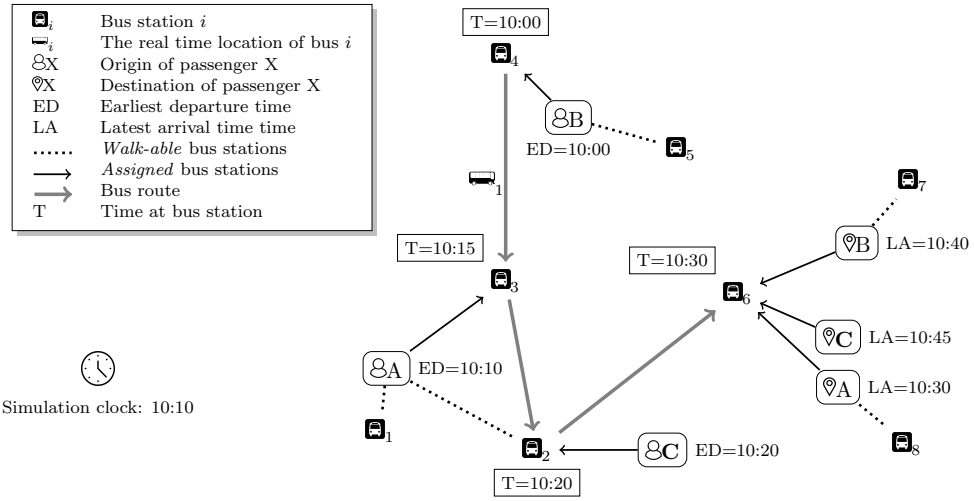
In previous work, Melis and Sörensen (2020) introduce the (static) on-demand bus routing problem (ODBRP) as a way to model a large scale on-demand bus system with bus station assignment. Given are a set of bus station locations, a set of identical buses with fixed capacity, and a set of transportation requests. A request consists of a set of possible origin/departure and arrival/destination stations, as well as a time window (an earliest departure and latest arrival time). The earliest departure time is the earliest time a passenger can be picked up at one of the possible departure stations. Similarly a passenger has to arrive at one of the possible arrival stations before the latest



(a) Instance and static solution



(b) Dynamic request of passenger C pops up



(c) Dynamic request of passenger C is added to the solution

Figure 1: Example from static ODBRP to real-time ODBRP: insertion of dynamic request

arrival time. The transportation needs to happen within this time window. The possible stations for departure and arrival are within walking distance from the passenger's actual departure and arrival location. In the ODBRP two decisions have to be made simultaneously (1) the assignment of customers to departure and arrival stations and (2) the routing of the buses. The first, *bus station assignment*, chooses one of the possible departure bus stations, to which the passenger is guided for the pick-up and one of the possible arrival stations, from which the passenger can walk to the final arrival location. This gives more flexibility to the routing, as customers are pooled and buses avoid to make unnecessary stops. The objective is to minimize the total user ride time (URT) or the time passengers spend on the bus.

An example can be found in Fig. 1a. The figure shows a solution based on two requests: passenger A and passenger B. To clarify, passenger A can only leave at one of the possible origin bus stations (namely station 2 or 3) after 10h10, which is the earliest departure time. Passenger A has to arrive at one of the possible destination bus stations (stations 6 or 8) before 10h30, which is the latest arrival time. A bus route is created (and bus station assignment is performed) to first pick up passenger B at 10h, then pick up passenger A at 10h15 and to drop them off at station 6 at 10h25. The time needed for passengers to get on/off the bus, when the bus is standing still at a bus station, is called the dwell time, but is not currently represented in the example for simplicity reasons. The depicted solution is feasible as all passengers are transported within their time window. These requests are scheduled before the departure of the buses, for example, one day ahead of time and are called static requests.

This work continues on the same problem, but introduces and analyzes the *real-time* ODBRP. An instance of the real-time ODBRP consists of a set of known (static) transportation requests, a set of dynamic transportation requests, a set of bus station locations and a fleet size of buses with a fixed capacity. Contrarily to the static requests, the dynamic requests are not known in advance and become known gradually when the buses have already started to work on picking up and dropping off the known requests. The departure location of dynamic requests will mostly be the current location of the customer. Each request from this set is sent either just-in-time, when the customer wants to leave as soon as possible, or a little while before desired departure. The time when a request is sent and appears in the system, is further called the issue time of a request. From this time on-wards, the request can be scheduled. Assuming the first bus starts his service at time zero, static requests have a negative issue time, while dynamic requests have a positive one. The time between the issue time and the desired departure is called the lead time. The application responds quickly with a transport option, telling the customers which bus station they have to walk to and which bus will pick them up. When confirmed, the system executes the request, picking up and dropping off the customers as promised.

The example continues in Fig. 1b showing the setting at 10h10, which is the issue time of the request of passenger C. A simulation clock depicts the current time. The bus is now located somewhere in between station 4 and station 3.

Passenger C wants to leave as soon as possible and therefore has an earliest departure time of 10h20. The ten minutes between the issue time and the earliest departure time are needed to walk to a bus station. The only possible bus station passenger C can walk to is station 2, this is indicated with the dotted line in the figure. The passenger wants to arrive at station 6 before 10h45, which is the latest arrival time. From this moment the algorithm will calculate where and how to insert this request in the previously defined bus routes. Figure 1c shows the insertion of this dynamic request. Instead of driving directly from station 3 to station 6, the bus makes a small detour to station 2 to pick up passenger C[1]. Afterwards, the bus drops everyone off at station 6. A small delay occurs for passengers A and B compared to the initial schedule, however there are no violations against the imposed time window. Every passenger arrives before/at the latest arrival time. Passengers will only know their actual departure/arrival times last minute. Consequently it is important as an operating company to make accurate estimations of the arrival times. This work is a first step in this direction.

Including the possibility of dynamic requests, requires a real-time or online re-optimization of a partially executed solution. Compared to the static case, the solution quality, expressed in the total URT, will decline. As this results in lower customer satisfaction, and consequently lower revenues for the operating on-demand bus company, we will call this decline *the cost of dynamic requests* and investigate its characteristics. More specifically, we will investigate how large the decline in URT is in general, and determine whether passengers that issue dynamic requests suffer a larger (or smaller) increase in URT than passengers that submit their request a longer time in advance.

The remainder of this paper is structured as follows. In Section 2 a literature review on mobility-on-demand (MOD) and on-demand public transportation can be found. In Section 3 a detailed description of the heuristic used to solve the real-time ODBRP is presented. In Section 4 we analyze the influence of bus station assignment and the

[1]One might wonder why station 3 is not simply skipped, because passenger A would also be able to walk to station 2. This is because at 10h10 passenger A would already reach station 3. There will be too little time left to redirect the walk to station 2 and be on time to catch the bus. This part of the route is presumed fixed (see Section 3)

difference between the static and dynamic version of the ODBRP. Furthermore, we identify "the cost" of dynamic requests. Finally, we conclude this paper in Section 5.

2 Literature review

A large scale dynamic on-demand public bus system can be categorized under the umbrella term mobility-on-demand (MOD). MOD offers a user a customized travel plan, in accordance with the actual demand of the customer, and this on a real-time basis. However, MOD not necessarily includes ride-sharing, i.e. a large number of MOD-systems only transport a single user at a time. Ride-sharing brings together customers with analogous journeys and time schedules (Agatz et al., 2012). For example, UberPOOL is based on ride-sharing, bringing together users to share the cost, but taking a personal Uber is not (Uber, 2019, 2016). MOD and ride-sharing systems can be reactive or proactive. Reactive systems answer to the demand as it comes and only take into account the current requests for transportation. The second considers along with the current, also future demand. When building routes for the currently known requests, anticipations are made to incorporate future requests. Examples of literature on proactive systems can be found in Tsao et al. (2019), Bruni et al. (2014) and van Engelen et al. (2018).

The MOD and ride-sharing systems are found to be especially useful in the context of autonomous vehicles. In the field of MOD systems, this is called autonomous mobility-on-demand or AMOD (Pavone, 2015; Tsao et al., 2019). It is found by Pavone (2015) that using AMOD instead of regular private car ownership reduces the total mobility cost with 46%. Winter et al. (2018) conclude that using autonomous vehicles for a public bus system has cost savings as a result of vanishing drivers' cost. The real-time ODBRP introduced in this paper, can both be implemented with autonomous as non-autonomous vehicles and is implemented as a reactive large-scale ride-sharing (A)MOD system. Proactive models are left for future research.

Other advantages of ride-sharing MOD systems are that it is preferred over traditional public transport because of increased accessibility, causes lower user ride times and decreased total distance driven, which in turn results in less congestion (Yan et al., 2019; Melis and Sörensen, 2020; Tsao et al., 2019).

ODBRP	**DARP**
Station-based + bus station assignment	Door-to-door[2]
One time window within which pickup and drop-off happens	Separate time window for pickup and drop-off
No maximum ride time constraint	Maximum ride time constraint
Quality-based objective: min. user ride time	Cost-based objective: min. total distance driven
High demand	Low demand

Table 1: Major differences between the ODBRP and the DARP

From a modeling perspective, an on-demand public bus system is most closely related to the dial-a-ride problem (DARP) (Cordeau and Laporte, 2007), however fundamental differences can be found. In the static DARP a set of requests for travel are given, with specific origin and destination locations (door-to-door), and vehicles need to be routed to serve these requests. In contrast to the ODBRP, there are separate time windows for the pick-up and drop-off of the request and a constraint on the maximum ride time, while mostly minimizing the cost or total distance driven. The DARP is classically applied for the transportation of the disabled, sick or elderly, but is more recently used to model transits from suburban areas to major transport hubs, thus working as a feeder system to regular public transport (Integrated DARP introduced by Häll et al. (2009); Posada et al. (2017); Stiglic et al. (2018)) or it is generally used in areas with low demand. This is opposed to on-demand public transport systems, which have better performance in high demand applications, especially when bus station assignment is applied to pool passengers (Archetti et al., 2018; Jokinen et al., 2011; Melis and Sörensen, 2020). All major differences between the ODBRP and the DARP are summarized in Table 1. A last difference between the ODBRP and the DARP (and other vehicle routing problems) lies in the fact that a bus is allowed to stop multiple times at the same bus station. The time windows and/or transportation directions of requests are substantially different from each other and sometimes cause a bus to stop at a certain bus station twice but in a different time and while driving in a different direction afterwards.

In reality a certain number of requests comes in just-in-time or a little while before departure. Therefore, the dynamic DARP is more realistic, compared to the static variant, but is not investigated as much. Reviews of the

[2]Only one work of the DARP using alternative nodes (a feature that can be compared to bus station assignment) was found in Brevet et al. (2019). Unlike in the ODBRP, the decision on the alternative nodes and the routing does not happen simultaneously.

dynamic DARP can be found in Molenbruch et al. (2017) and Ho et al. (2018).

In the DARP it is assumed that customers can be picked up/dropped off anywhere. While this might be possible in suburban areas or might be necessary for the transport of the disabled, this is not the most efficient way in an urban context. For safety reasons, buses must stop at clearly signposted locations, bus stations. The question rises whether or not to let the customers decide which bus stations they want to use for their pick up and drop off. Gökay et al. (2019) introduce a simulation of an online on-demand ride-sharing system with meeting points and conclude that by pooling pickups and deliveries of passengers in meeting points, the overall efficiency of the system increases. The success rate enhances and the vehicle costs decline. Czioska et al. (2019) investigate a static shared on-demand transport system with meeting points. After clustering similar requests based on origin and arrival location as well as desired departure time, they determine a meeting point for every cluster, which is a time-dependent hot spot. The number of necessary stops is reduced and users' satisfaction increased (measured by travel time, price and additional walking time). Koh et al. (2018) only allow minor flexibility concerning the pickup and drop off stations of passengers. They allow the algorithm to redirect a passenger to a nearby bus station, e.g., a bus station across the street, to increase efficiency. Based on this literature and the findings of Melis and Sörensen (2020) one could argue to include bus station assignment and let the algorithm decide where a passenger is picked up and dropped off.

It is only in the last couple of years that research is done to optimize *large-scale* dynamic reactive on-demand ride-sharing systems. However in general, literature on this topic is scarce and none of the papers found mention bus station assignment. Vallée et al. (2017) consider an online shared transportation system by solving the dynamic DARP and choose to maximize the number of served requests. They use a quick insertion heuristic every time a new request appears to give a quick answer to the customer about the feasibility of the request. Subsequently they use an adaptive large neighbourhood (ALNS) heuristic to optimize the solution. Their algorithm is able to cope with more than 2000 requests. However they just lead the customer to the nearest predefined station, instead of using bus station assignment. Alonso-Mora et al. (2017) solve an on-demand high capacity ride-sharing problem via dynamic trip vehicle assignment and constrained optimization. They try to re-localize empty vehicles to places with high demand. They test their algorithm on the New York City taxicab public dataset and find that even without ride-sharing (this means only one person per ride), they can decrease the number of taxis from 13.000 to 3000, again contributing to the fight against congestion. They use a door-to-door approach, without predefined stations. They conclude that by including high capacity ride-sharing, the service rate increased, while the waiting time and the total vehicle distance dropped. Also Jokinen et al. (2011) and Archetti et al. (2018) analyze the performance of dynamic large-scale ride-sharing or on-demand public transportation problems with a simulation study, by using a simple greedy insertion heuristic. They conclude that a mass demand responsive system would not make other transport options obsolete, but is still efficient and results in cost and travel time savings compared to a conventional bus system. Additionally it would help to solve the pollution and congestion problems. Winter et al. (2018) simulate such a system with automated vehicles and also find that the simulated on-demand system had similar costs as the existing fixed bus system, but the average waiting time is less.

In this paper, we extend the existing literature by introducing a heuristic for the real-time ODBRP, an online on-demand bus system *with bus station assignment*. Furthermore, the increases and variety of the total user ride times caused by dealing with dynamic requests compared to only dealing with requests known in advance, have not yet been investigated. Smaller total user ride times reflect the quality of the service and have a direct positive impact on the willingness to pay (WTP) of passengers for the on-demand bus service. Brownstone et al. (2003) find that the WTP to reduce commute time is roughly $30 per hour. Time has a monetary value, consequently on-demand bus operators can ask differentiated prices to their customers depending on the URT. Furthermore, Li et al. (2010) investigate the WTP for travel time reliability and find through empirical evidence that both the reliability of travel times and the travel time savings have a positive impact on the WTP. Ergo, dynamic requests impose a cost for the operator, as longer and more unpredictable URT's cause the WTP and prices to drop. Investigating the cost of dynamic requests is therefore of great importance for operating on-demand bus companies.

3 Heuristic

When solving the real-time ODBRP, first a solution is created with the static set of transportation requests (since these are the only requests known at design-time). This is called the static part of the algorithm. Afterwards, the partially executed solution is adapted when the dynamic requests are issued, which is the dynamic part of the algorithm. In Melis and Sörensen (2020) a Large Neighborhood Search (LNS) heuristic is used to solve the static ODBRP. As the real-time ODBRP is an extension of the static variant, this heuristic is used as a first building block to solve the real-time ODBRP. We included Algorithm 1 in Appendix A for an overview of the entire heuristic, both

the static and the dynamic part.

The goal is to minimize the total user ride time (URT), which is the time the passengers spend on the buses. One could also choose to minimize the total travel time, which includes the walking time in between the actual origin/arrival location and the assigned origin/arrival station. However, it is chosen not to optimize walking times as walking preferences and speeds differ between people. Additionally, optimizing the URT is the most logical option from the bus company's point of view. Instead, a constraint is set on the maximum walking time, making it impossible for walking times to become too long. In this work, we impose a maximum walking time of ten minutes for every request, but this can easily be altered and even personalized depending on the request.

3.1 Static part

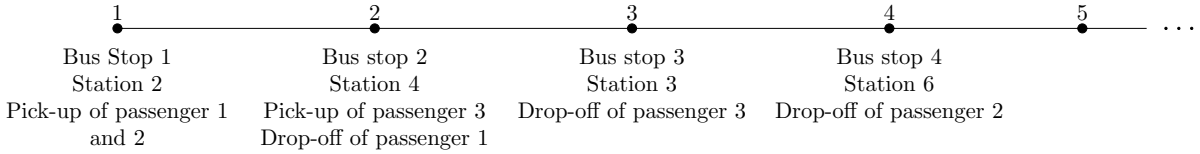


Figure 2: A bus route in the ODBRP

In the ODBRP, a route consists of a sequence of bus stops as is shown in Fig. 2. Each bus stop occurs at a bus station, which is a physical location where passengers are allowed to get on or off the bus.

In a nutshell, the static heuristic starts by constructing an initial solution with a greedy heuristic, which inserts requests where the total URT increases the least. When inserting a request the increase in total URT is calculated as the URT of the newly added request plus the increase in URT for the existing passengers in the route, caused by the insertion of the new request. Once every request has been inserted in this initial solution, *a static optimization phase* starts, consisting of several "static iterations". Every static iteration normally consists of two parts: a destroy-and-repair and a local search (LS) part. At every iteration the destroy-and-repair part clears some bus route(s) entirely, and re-adds the requests, which were located in these bus route(s), in the same greedy manner as when constructing the initial solution. The number of bus routes to clear is decided as a uniformly distributed random number between 1 and 5% of the total fleet size. This number is deliberately low to keep the most efficient bus routes in the solution. The bus routes to clear for the destructive operator are chosen according to a criterion, randomly selected among one of the following three criteria: largest total URT, largest URT to distance ratio and smallest number of passengers served in the route. When *all* requests are re-added, within the same iteration, local search (LS) takes every request out one by one and tries to add it in another position either in the same route or in another route, to further minimize the URT. When, for five consecutive iterations of destroying and repairing the solution, *not* all passengers can be re-added, the objective function changes briefly to maximizing the number of requests served, until every request is served again (which restores feasibility). The objective function change is done through a LNS-operator change. For the destroy operators the new focus lies on making space in bus routes to schedule additional passengers. The bus routes selected to be cleared are chosen according to three other criteria: random, smallest number of passengers served, and maximum amount of waiting time. Waiting time is the time the bus is standing still at a certain stop to wait for other passengers to get on. The repair operator, on the other hand, inserts new requests where the route extension in distance or driving time is the smallest. This way, buses create time to serve extra passengers. When all passengers are scheduled once again, the objective changes back to minimizing the total URT. For a more detailed explanation we refer once again to Melis and Sörensen (2020).

3.2 Dynamic part

After the static part of the algorithm, the dynamic part of the problem begins. Requests that have a strictly positive issue time are revealed as the simulation clock moves forward and are scheduled one by one in the different bus routes. The set of dynamic requests is arranged by their earliest departure time in increasing order. The simulation clock, indicating the current time, is constantly tracked, and is equal to the earliest departure time of the first new request minus the lead time and the ten minute walk. The algorithm tries to insert the dynamic request in the solution with the same constructive heuristic which is used in the static part, however taking into account that a part of the route cannot be changed anymore. Also, the delay caused by the insertion of the new request can not induce the arrival

time of the already scheduled passengers to exceed their latest arrival times as this is the service promise that was made.

To determine which part of the route cannot be changed any more, because its associated events have already been (at least partially) executed, a *locking point* is introduced. This is the point in the route before which all the stops are fixed (in general, the locking point is defined as is described in Algorithm 2 Appendix A). No route extensions or extra pick ups can be scheduled before the locking point. After this point, everything can be changed. There are two basic rules to fix a bus stop. Logically no past events can be modified. Stops where passengers get on or off the bus in the past, can not be changed. When a passenger both gets on and off in the past, there is no need for change, but when a passenger gets on the bus in the past and is scheduled to get off in the future, the arrival stop of this passenger can still be altered. The second rule is to fix all bus stops to which vehicles or passengers are moving to at the current time. The stop to which a bus is currently driving, is presumed fixed. It would be unfair to the passengers scheduled to get off the bus at the next bus stop, to change this, as they are reaching their destination. Stops which passengers are currently walking to, are fixed as well. When passengers are walking to a bus station, they are about to be picked up. If a bus stop executing a pick up event is changed last minute, passengers might not have enough time to walk to another bus station or they might have to wait a long time at the station because other passengers are scheduled before them at the latest possible moment. In the last case, they would probably prefer to have waited at home. To illustrate, the locking point in Fig. 1b is at the second stop. A final remark is that stops are always fixed in a consecutive manner. If for example the first and third stop need to be fixed, the second stop is fixed automatically. Nothing can change concerning a fixed stop, neither the pick up and drop off events as the time of arrival and departure of the bus at the stop.

It should be noted that when a dynamic request comes in just-in-time, the passenger is not necessarily scheduled to depart right away, at least if the time window allows this. For example, a dynamic request might come in at 8 a.m. with a time window between 8 and 9 a.m. After initial insertion, the request is scheduled to be picked up at 8h30. The passenger starts walking at 8h20. This means that between 8 and 8h20 a.m., the route of this passenger can still be adapted, even though the request was send in real-time. Of course, once a request is accepted, the algorithm cannot delete it from the schedule.

Table 2: Toy example - Requests and distance matrix

Static requests	P	e_p^u	l_p^o	Departure stations	Arrival stations
	1	40	100	1, 2, or 3	5 or 6
	2	30	130	7, 8, or 9	10 or 11
	3	90	220	11	4
Dynamic requests	P	e_p^u	l_p^o	Departure stations	Arrival stations
	4	60	150	4	10 or 11

Distance Matrix

	1	2	3	4	5	6	7	8	9	10	11
1	0	5	5	20	15	13	20	25	20	35	35
2		0	5	20	15	14	15	15	20	35	35
3			0	20	15	10	12	14	14	25	30
4				0	30	35	40	40	40	60	60
5					0	5	30	28	32	40	41
6						0	30	28	32	40	38
7							0	5	5	40	45
8								0	5	45	50
9									0	45	50
10										0	5
11											0

P = Passenger, e_p^u = Earliest departure time of P, l_p^o = Latest arrival time of P

3.2.1 Toy example

Inserting a dynamic request To illustrate the adaptations made to the static heuristic, a toy example is used. Table 2 shows three static and one dynamic request. These requests, together with a distance matrix between the possible stops (see Table 2), a fleet size and given capacity of the vehicles form the instance. Assuming there is only one available bus with capacity 8, the static requests are already scheduled in this vehicle with the static part of the heuristic. A dwell time of one time unit is adopted. The scheduled route is shown in Fig. 3a. The bus departs, but the route can still be altered when new requests come in. In this example, the (only) dynamic request (P4) is issued at time 50, which is ten minutes before the earliest departure time, because P4 needs time to walk to a bus station. P4 has to get on the bus at station 4 and arrive at one of the destination stations (10 or 11) before time 150. The current time is 50, therefore stops 1 and 2 in the scheduled bus route are fixed as they are scheduled in the past. However, we assume stop 3 to be fixed as well. The rationale behind this is that the bus is already on its way to station 6, the third stop. Furthermore, the algorithm checks if there are passengers already walking to one of the scheduled bus stops in the route. To check this, the algorithm looks for the first stop further in the route where passengers are scheduled to get on the bus. If such a stop exists, it is checked if any passengers are already walking to this stop. In our example, up until the third stop is presumed fixed. In the fourth stop, P3 is scheduled to get on the bus. The departure time of the fourth stop minus the stop time minus the maximum walking time can not be smaller than the current time, as this would mean that this particular passenger might be already walking to the assigned departure stop. The stop time is deducted because this is the time needed for the passenger to actually get on the bus before it can leave. In the toy example this would be $96 - 1 - 10$, which equals 85. This is after the current time, meaning that the fourth stop does not need to be fixed and the locking point is at the third stop.

The (only possible) insertion of the dynamic request of passenger 4 is shown in Fig. 3b. The stops pictured in gray are presumed fixed. Every stop after this point can still be altered or delayed as long as it meets the constraints for the existing passengers in the route. The stop with the asterisk is the insertion of the origin of P4. For the arrival no extra stop needed to be created as a stop at bus station 11 was already scheduled in the solution.

Bus stop	1	2	3	4	5
Bus station	7	3	6	11	4
Arrival time	30	43	54	95	155
Departure time	31	44	55	96	156
P on	2	1		3	
P off			1	2	3

(a) Static solution

Bus stop	1	2	3	*	4	5
Bus station	7	3	6	4	11	4
Arrival time	30	43	54	90	150	211
Departure time	31	44	55	91	151	212
P on	2	1		**4**	3	
P off			1		2, **4**	3

(b) Dynamic request insertion with locking point at the third stop (gray area is presumed fixed)

Bus stop	1	2	3
Bus station	7	3	6
Arrival time	30	43	54
Departure time	31	44	55
P on	2	1	
P off			1

(c) Route cleared by LNS destroy operator

Bus stop	1	2	3	4
Bus station	7	3	6	11
Arrival time	30	43	54	95
Departure time	31	44	55	96
P on	2	1		
P off			1	2

(d) Route made feasible after LNS destroy operator

Figure 3: Toy example

Large Neighborhood Search If the insertion of the dynamic request succeeds, the algorithm continues with some iterations of large neighborhood search (LNS) (destroy-and-repair operations and possibly some LS), however the necessary adaptations are once again made for dealing with dynamic requests. In the static part the destroy operators in the LNS clear entire bus routes. In a dynamic environment this is not possible as a part of the route is fixed. To determine which part of the route is fixed, the same rules as explained in Section 3.2 stand. The remainder of the route is cleared, meaning that all stops after the locking point are deleted from the route. For example, if the route depicted in the toy example in Fig. 3b, would be chosen by the LNS heuristic to be cleared, the result would look like the route pictured in Fig. 3c. However, it should be noted that this is an infeasible solution, as P2 gets on the bus at the first stop, but is not scheduled to get off the bus later on. The in-feasibility is first restored in a greedy manner. In the toy example P2 is the only passenger that needs a newly scheduled arrival stop. P2 has two possible arrival stations: 10 and 11. The last stop in the itinerary of the route in Fig. 3c is station 6. The station inducing the least increase in total user ride time is chosen. In this case this is station 11 with a distance of 38 from station 6 (compared to a distance of 40 when choosing station 10). As a consequence station 11 is chosen as the arrival station of P2 and is added to the route as shown in Fig. 3d. If there are multiple passengers without an arrival stop after the route is cleared, the order in which they get off the bus in the current route is saved and used as the order to add their arrival stations again in the route after the clearing. In addition, if passengers are previously scheduled together in the same stop, this is done again to make sure the passenger pooling is not lost. In most cases this would cause the solution to be feasible, meaning that all passengers get to their arrival stations before their latest arrival time. In

rare circumstances, illustrated in Fig. B1 in Appendix B, this technique does not work. Therefore a feasibility check is performed after rebuilding the arrival stops. If not feasible, other possible arrival stations of these passengers are tested until a feasible solution for all passengers is found.

Local Search If all passengers are still scheduled in the given bus fleet after successfully inserting the request and after the destroy and repair operators, local search (LS) is performed. If both the departure and arrival of a request is scheduled in the future, if the bus is not currently driving to the departure stop of this request, and if the passenger of this request is not currently walking to the assigned departure stop, the request is entirely flexible for changes. LS takes the request out of the solution and uses the greedy heuristic of the static phase to attempt to reinsert it in a position where it minimizes the total URT. When re-adding the request, the same rules of fixing stops apply as before. Compared to the static LS, one additional change is made. If a passenger is already on the bus, the assigned arrival stop can still be altered. At least if the bus is not already driving to this stop and if there are no passengers currently walking to get on the bus at this stop. In this case, LS is done only on the level of the arrival. The scheduled arrival is taken out of the solution and the algorithm tries to reinsert it possibly somewhere better, but in the same bus. To illustrate, the toy example continues. After the destroy and repair operators, a fifth passenger is added to the route shown in Fig. 3d. P5 is, for example, taken out of another bus and added to this one because it lowered the total URT. The departure needs to be from station 1 and for arrival there are two possible stations: 10 and 11. The earliest departure time is 60 and latest arrival time is 120. Figure 4a shows the route after adding P5. Again, the gray area is considered fixed. The total URT of this route equals $10+73+41$ or 124. However, improvement is possible on the arrival level because P5 can have both station 10 and station 11 as a destination. By changing the arrival to station 10, as is shown in Fig. 4b, the total URT of this route lowers to $10+73+35$ or 118.

Restoring feasibility If the algorithm directly fails to insert the dynamic request in the solution, meaning that no feasible solution like the one in Fig. 3b is found, the same objective function change as in the static part occurs. The algorithm uses the same destroy and repair operators that maximize the number of requests served to make room to add this new request. Of course the same adaptations concerning the fixed parts of the routes are made. The algorithm keeps clearing and repairing bus routes until the dynamic request is successfully added to the solution or until a stop criterion is met. For more information on the stop criterion we refer to Section 3.4.2. If, because of the objective function change, the request is inserted successfully, the algorithm continues with 20 iterations of local search to quickly restore the solution quality which worsened by not focusing on minimizing the total URT. Otherwise, the algorithm goes back to the solution without the new dynamic request. Note that only new dynamic requests can be denied, static requests or requests promised to be served will never be removed from the schedule to be able to serve new requests.

Bus stop	1	2	3	4	5	6
Bus station	7	3	6	1	11	10
Arrival time	30	43	54	68	104	110
Departure time	31	44	55	69	105	111
P on	2	1		5		
P off			1		2	5

(a) A fifth passenger is added to the solution after the destroy and repair operations

Bus stop	1	2	3	4	5
Bus station	7	3	6	1	10
Arrival time	30	43	54	68	104
Departure time	31	44	55	69	105
P on	2	1		5	
P off			1		2, 5

(b) Local search on arrival level

Figure 4: Toy example: a fifth passenger is added to the solution and local search is performed to improve the solution

3.3 Instance generation

The instances used in this work are randomly generated. Passengers have an earliest departure time between minute 10 and 70 of the time continuum. The latest arrival time of a request is equal to the earliest departure time plus two times the (Euclidean) distance that needs to be traveled (in minutes) plus twenty minutes. The fixed term of twenty minutes is chosen to make sure that requests with short distances still have a reasonable time window. The maximum walking time to the bus stations is set to ten minutes. The 121 bus stations are equally distributed in a 100×100 Euclidean plane. From the total set of requests, the dynamic requests are randomly drawn. The partition static and dynamic requests is decided upon in advance. For research purposes, we assume all dynamic requests have the same lead time, of course this is easily changeable in the instance generator. The assumed lead time for dynamic requests is set at zero, unless stated otherwise (the requests are issued just-in-time). In all our experiments, the capacity of the buses is set at 8 and the fleet size is variable. Results shown on the graphs in the following sections are averages of ten different instances, where each instance is run three times. The set of dynamic requests is considered as given when rerunning the same instance three times.

3.4 Algorithm analysis and calibration

In this section the algorithm described in Section 3 is analyzed. First, we determine how many static or dynamic iterations are necessary in Section 3.4.1. Afterwards, in Section 3.4.2 it is investigated what stop criterion is best used after an objective function change. As is explained in Section 3 the change is called when the algorithm has failed to insert a dynamic request, but cannot last too long as the potential future passengers needs a fast response.

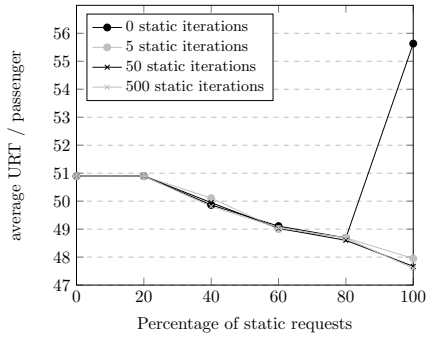
3.4.1 The importance of static and dynamic optimization

In this section, the importance of static and dynamic optimization investigated. We want to determine how many static and dynamic iterations are necessary to find good solutions and examine which type of optimization has the most influence on the solution quality. The importance of static optimization is expressed in the number of static iterations performed. These iterations are performed on the requests known beforehand and are therefore executed before the buses actually depart. The number of iterations can be large as time is not an issue. Static optimization causes the initial solution of the dynamic optimization to be of better quality. Dynamic iterations are performed every time a new request becomes known to the system. Consequently the dynamic iterations need to be performed fast and need to be substantially lower than the static iterations. One iteration consists of one round of LNS and LS. We will examine to what extent a good initial solution for the dynamic part is important and also analyze the importance of the dynamic optimization.

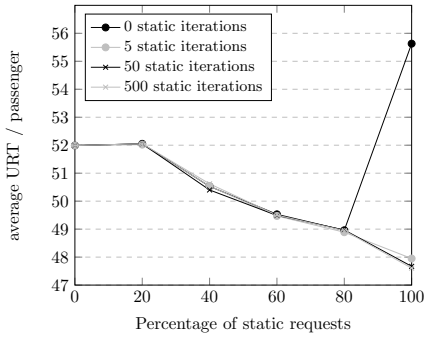
First, the algorithm is executed when all requests are known beforehand, this is the purely static ODBRP. Then, the same instances are solved but with only a fraction of the requests known beforehand. The influence of the static iterations is examined in Fig. 5a. On the x-axis the percentage of static requests is plotted, this is the degree of dynamism of the problem. The lower the percentage of static requests, the higher the level of dynamism. On the y-axis the average URT per passenger can be found. The number of dynamic iterations is set relatively high at 15 iterations for each dynamic request. The figure shows that the number of static iterations solely makes a large difference in the purely static ODBRP. In this case, a huge drop in total URT per passenger is seen between 0 or 5 static iterations and again a significant drop between 5 and 50 iterations. The difference in average URT per passenger between 50 and 500 static iterations is almost nonexistent. When the problem is somewhat dynamic, no significant differences can be found (p-values of the two-tailed paired t-test between 0.16 and 0.45), indicating that static optimization is of less importance. Even in the completely static scenario, only 5 to 50 iterations suffice to obtain a good solution quality.

In Fig. 5b the same experiment is executed, but the number of dynamic iterations is set at 1 iteration per dynamic request. Even with this small number of dynamic iterations, a similar conclusion can be drawn: static iterations are of minor importance in the real-time ODBRP. Contrarily a quick comparison between Fig. 5a and Fig. 5b shows that dynamic iterations are relevant and lower the objective function value.

To further demonstrate the influence of the dynamic iterations, Fig. 6a shows the average URT per passenger for different percentages of static requests, using the maximum value of 500 static iterations and for different numbers of dynamic iterations. The graph shows clearly that the number of dynamic iterations influences the outcome more when the percentage of static requests is relatively low. From a percentage of 40 onward the differences between using 15, 5 or 1 dynamic iteration(s), become smaller. When the number of static iterations decreases, these differences remain visible. This is shown in Fig. 6b. In this graph the number of static iterations is kept at zero. As was concluded from Fig. 5a and Fig. 5b, not doing static optimization has especially consequences in the purely static



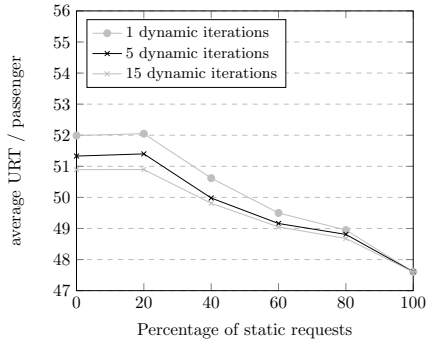
(a) Using 15 dynamic iterations per dynamic request



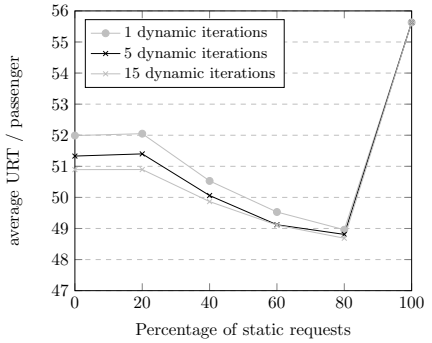
(b) Using 1 dynamic iteration per dynamic request

Figure 5: The effect of static iterations on the average URT per passenger for instances with 2000 requests. A fleet size of 500 is chosen to make sure everyone can certainly be served.

ODBRP. Furthermore it causes the average URT of the initial static schedule to be relatively high. The average URT per passenger of the static requests decreases 10.07% on average when starting from a non-optimized static schedule and doing one dynamic iteration per dynamic request. This percentage increases to 10.47% when doing 15 dynamic iterations per dynamic request.



(a) Using 500 static iterations



(b) Using 0 static iterations

Figure 6: The effect of dynamic iterations on the total URT per passenger for instances with 2000 requests. A fleet size of 500 is chosen to make sure everyone can certainly be served.

To conclude this section one could argue not to invest too much in static optimization, especially when the on-demand system used is highly dynamic. In this case, it is better to use the time in between dynamic requests to further optimize the solution. However, when the bus system has a more static nature, e.g., a bus service for employees of different companies using an on-demand bus system where they can easily reserve a seat a couple of days in advance, but might have the opportunity to book a last minute seat by exception; it is still extremely important to invest time in static optimization. Also, when it is highly uncertain if dynamic requests will come in, the safer option would be to do some static optimization as well. This would cause minor inconvenience as requests are known in advance, so time is not an issue. On the other hand, the number of dynamic iterations does have a large influence on the solution quality. For medium degrees of dynamism the difference between doing 5 or 15 dynamic iterations is rather small, but for a highly dynamic on-demand bus system, the maximum number of dynamic iterations is recommended. However this number will still have to be rather small, to provide a quick response to passengers who have submitted a dynamic request. Because of this, and for equal comparisons in further analyses, we continue to work with 500 static iterations and 15 dynamic iterations per dynamic request.

3.4.2 Stop criterion objective function change

In Algorithm 1 in Appendix A and as mentioned in Section 3, it is indicated that when a dynamic request can not be inserted initially, an objective function change is performed. Until every request can be served or until a stop criterion is met, destroy and repair operators that maximize the number of passengers served, are carried out. In reality the customer that has sent this dynamic request wants to know whether or not the request can be served as soon as possible. Therefore the stop criterion has to assure a short response time. A maximum response time of one minute is applied. For instances with 2000 requests, one minute computation time would amount to circa 16.000 iterations of the LNS maximizing the number of passengers served. However, in reality we saw that this large number of iterations is not necessary, because if the heuristic finds a feasible solution for the request to be inserted, this typically happens quickly. Of course this depends on the difficulty of the instance: the total number of requests, the percentage of static requests and the fleet size. It is more challenging to schedule a large number of requests that come in just-in-time in a relative small number of buses. Figure 7 illustrates this finding.

The number of iterations that are needed in order to find a feasible insertion for the dynamic request is plotted against the percentage of static requests for different fleet sizes. A maximum fleet size of 400 is chosen because this is the smallest fleet size with which every request can be served for all levels of dynamism. A fleet size lower than 300 is not adopted, because in this work we aim at serving every request and minimizing the total user ride time. For example, using a fleet size of 250, the system would only be able to serve circa 1900 requests in the purely dynamic ODBRP. For a fleet size down to 350, for all levels of dynamism, the number of iterations necessary lies below 200. Only for the fleet size of 300 this number increases drastically up to 1035 iterations on average for the completely dynamic problem. Therefore we chose to set the stop criterion at 5000 iterations. This way a decent buffer is adopted, while ensuring that the customer receives a fast response. Additionally, it should be noted that the lower the level of dynamism the easier it is for the algorithm to find a successful insertion, thus the faster the sender of a dynamic request will have an answer to whether of not the system can carry out the request. Of course when the fleet size is large enough relative to the number of requests the objective function change is barely needed.

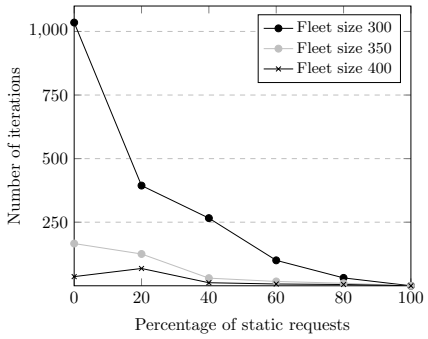


Figure 7: The average number of iterations necessary to successfully insert a dynamic request, which initially failed to be inserted, using the LNS which maximizes the number of served requests, for instances with 2000 requests and different fleet sizes

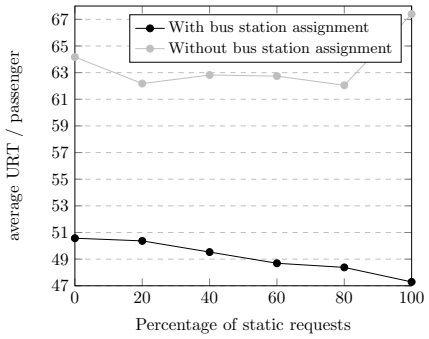


Figure 8: The influence of bus station assignment for different degrees of dynamism for instances with 2000 requests and a fleet size of 500

4 Results analysis

4.1 The influence of bus station assignment

In this section the influence of bus station assignment is investigated for different degrees of dynamism. Without bus station assignment, passengers choose their own bus stations for departure and arrival, normally the ones closest to their actual departure and arrival location. With bus station assignment, the algorithm decides which bus stations are chosen within a certain walking distance from the actual departure and arrival location. As was concluded in Melis and Sörensen (2020), bus station assignment causes the average URT per passenger to drop significantly for the

static problem. The results shown in Fig. 8 show that the same is true for the dynamic problem. On average, over all levels of dynamism, the average URT per passenger lowers 21.7% by including bus station assignment. For the static variant of the problem this difference is even 30.8%. This percentage increase can be explained by the need for more (static) iterations to lower the total URT. Without bus station assignment it is harder to pool similar requests and a larger number of stops need to be made.

4.2 The cost of dynamic requests

Requests with a positive issue time are expected to cause an increase in the average URT compared to the situation in which all requests are known before the optimization starts. As was mentioned in Section 2, longer and more variable URT's cause the willingness to pay of passengers to drop. In our experiments, this cost is defined as the deterioration of the objective function value, namely the total user ride time (URT), caused by inserting requests in real-time instead of scheduling these *same* requests in a static manner, before the planning horizon starts. In Fig. 9 the effect of the percentage of static requests, thus the effect of the degree of dynamism, on the total URT per passenger is plotted for instances with 500, 1000, 1500 and 2000 requests to investigate to what extent the degree of dynamism has an influence on the total URT. To start, the fleet size is set by a relatively large value, but proportional to the number of requests, i.e. 125, 250, 375 and 500 respectively, to assure every request can certainly be served.

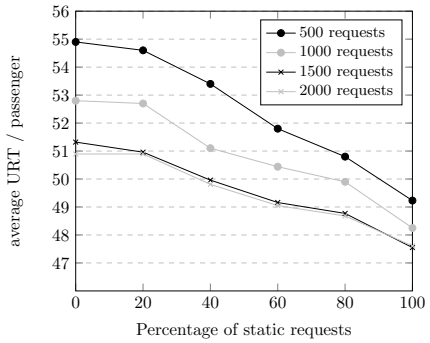


Figure 9: The effect of the percentage of static requests on the average URT per passenger

In general, a quasi linear trend can be seen when going from a percentage of 0% static requests to 100%. As a consequence it can be concluded that the more requests one knows before the buses depart, the better. For instances with 2000 requests there is a 6.7% difference in URT when going from a static to a fully dynamic system. For instances with 500 requests, there is a 11.5% difference.

Also the average URT per passenger becomes smaller the larger the number of requests, even though the fleet size grows in the same proportion. This can be explained by the economies of scale linked to the ODBRP. If there are more requests, it is more likely that some of these requests are very similar and can be scheduled easily together in the same bus without experiencing tremendous delays for picking up extra passengers. The average URT per passenger for instances with 1500 or 2000 requests is similar however, indicating that the economies of scale are not infinite.

The cost of including the possibility of dynamic requests, instead of only allowing static requests, is further investigated in this section. First, the effect of dynamic requests on the fleet size necessary to serve every request is analyzed in Section 4.2.1 as a higher fleet size also inflict a higher cost for the operator. Because of dynamic requests, delays occur for already scheduled requests in the solution. In Section 4.2.2 it is investigated who is affected the most by these delays: the passengers that send a request beforehand, or passengers who send a dynamic request. Also, the severity of the delays is evaluated to investigate the reliability/stability of the URT's of the initial solution. In Section 4.2.3 the effect of increasing the lead times is considered.

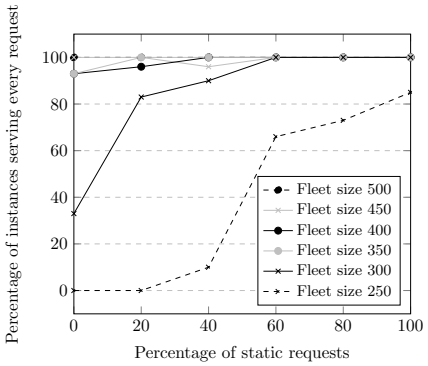
4.2.1 Is a fleet size increase necessary to incorporate the possibility of dynamic requests?

Adding the possibility of dynamic requests has its consequences on the fleet size necessary to serve every request and, procuring and maintaining a fleet of buses is a costly matter. Even with the objective function change referred to in Section 3, it becomes difficult to serve every customer with a limited fleet size of buses in a highly dynamic environment. To illustrate this, we lower the fleet size of instances with 2000 requests down to 250 instead of 500.

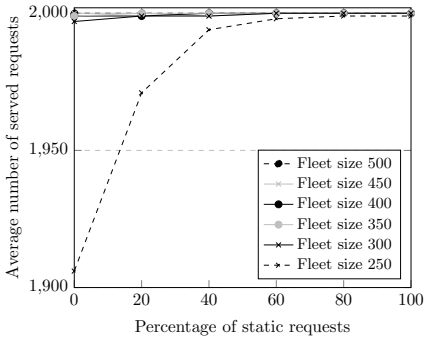
The capacity of the buses remains 8. Instances with 2000 requests were chosen because these give the most realistic picture. The graph in Fig. 10a shows the influence of the percentage of static requests on the ability of the heuristic to serve everyone with a given fleet size. Note that once again the same requests are used, independent of the degree of dynamism. It is only the issue times of the requests that differ. It is evident that an increasing level of dynamism causes difficulties scheduling every request, especially for smaller fleet sizes. For example, when there are 300 buses available, the algorithm has no trouble scheduling all requests when more than 60% of the requests are static. However if this percentage lowers, it is not possible to schedule every request in the solution. In fact, the algorithm is only able to schedule everyone in 33% of the instances when the problem is completely dynamic. Figure 10b shows for the same fleet sizes and levels of dynamism the average number of requests that could be served. Slight declines are visible for the completely dynamic problem for fleet sizes down to 300 buses. When the fleet size equals 250, the decline is larger. The algorithm can schedule on average 1906 requests in the purely dynamic ODBRP.

Besides the fact that the algorithm has trouble scheduling all requests given a relatively small fleet size and a high degree of dynamism, the average URT per passenger increases significantly in these cases. This is illustrated in Fig. 10c. For fleet sizes down to 500 every request can easily be served. Declines in URT can still be obtained by increasing the fleet size, with differences especially visible for high degrees of dynamism. When the percentage of static requests increases, this difference diminishes. From 450 buses downwards, the algorithm starts to have some difficulties planning every request. In the purely dynamic scenario on average 1999 of the 2000 requests are scheduled. Although it should be noted that the average URT per passenger increases substantially, again specifically for the completely dynamic ODBRP. The more restricted the fleet size, the larger the differences become. When going from 500 buses to 450, the difference is 6%, from 400 to 350 buses the divergence increases to 9%. In contrast, when adding buses to a fleet size which already serves everyone, the difference is a steady 3% at every fleet size increase of 50 buses (tested up until 600 buses).

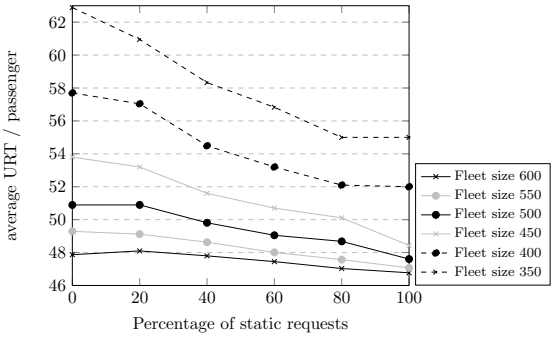
To conclude, an on-demand bus operator needs a larger fleet size when adding the possibility of dynamic requests to ensure that URT's do not become too large and to be able to serve both the static and the dynamic requests.



(a) The effect of the percentage of static requests on the percentage of instances able to serve every request when lowering the fleet size



(b) The effect on the number of requests served when lowering the fleet size



(c) The effect on the average URT per passenger for different fleet sizes (instances with 2000 requests)

Figure 10: The effect of the percentage of static requests on the solution quality for different fleet sizes

4.2.2 Who is affected by the dynamic requests and to what extent?

By adding the possibility for dynamic requests to come in, the already scheduled requests will mostly incur a delay compared to their initial schedule, as was shown in the example in Fig. 1b and Fig. 1c in Section 1. URT's of already scheduled requests become variable when dynamic requests are inserted in the solution and optimization procedures are performed during the planning horizon. The initially promised URT and arrival times at the issue time of requests may differ from the actual ones. In contrast, when the difference between the promised URT at issue time and the actual URT is small, the URT of this request is called a stable URT[3]. In reality, an on-demand bus operating company should be able to estimate the actual arrival times of passengers to include a buffer in the promised URT to make sure URT's become more stable.

This work is a first step towards this estimation by investigating the magnitude of this delay and examining to what extent the static or dynamic requests incorporate this delay. In general, the initially scheduled requests, the static requests, undergo the most optimization iterations and have the lowest URT per passenger. This is illustrated in Fig. 11. The average URT per passenger is plotted for several percentages of static requests, next to the average URT per passenger for only the static and dynamic requests respectively. Of course, when the problem is purely static (dynamic), the average URT per passenger equals the average URT per passenger for the static (dynamic) requests. For the different degrees of dynamism in between, it is clearly visible that the static requests have better URT's compared to the dynamic requests. In fact, the gap between the URT of the static and dynamic requests is relatively stable between a percentage of 20 and 60 static requests. In this case there is a 3.5% difference. In contrast, there is a 4.6% gap in favor of the static requests when the problem is barely dynamic (80% static requests).

[3]In a static scenario, URT's are completely stable, as the promised URT's are communicated a little while before the departure of the first bus and will not change anymore afterwards. Inserting requests and optimization procedures are all performed before communicating URT's to passengers.

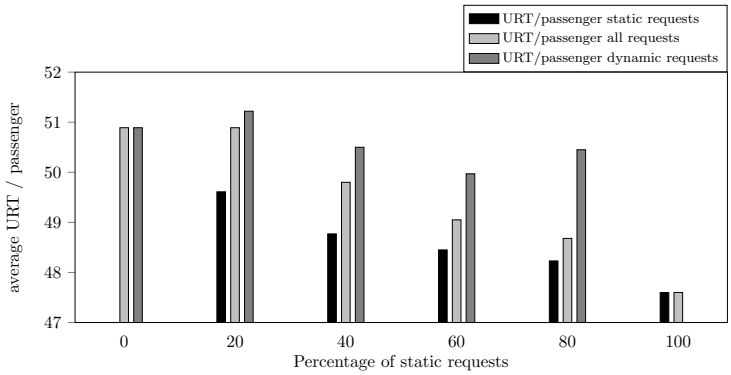


Figure 11: Comparison average URT per passenger static versus dynamic requests for instances with 2000 requests and a fleet size of 500 buses

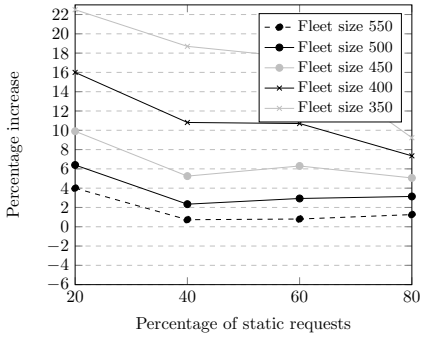
However, even though the static requests have lower average URT's compared to the dynamic ones, their URT increases significantly compared to the initial solution. Figure 12a plots the percentage difference in average URT per passenger for the static requests for different levels of dynamism and different fleet sizes. It is clearly visible that the lower the fleet size the more the URT of the static requests increases compared to the initially scheduled solution after the static optimization. The URT's are not stable, they increase during the planning horizon. Note that 500 static iterations are used to make sure the static solution has a decent quality as well. The increase in URT for the static requests is equally high for medium levels of dynamism (40 and 60% static requests), but larger when the problem is mostly dynamic (20% static requests) and smaller when the problem is mostly static (80% static requests). The last is also visible in the larger gap near 80% static requests in Fig. 11. For stable URT's of the static requests the fleet size needs to be large enough. When the fleet size is rather small and becomes even smaller compared to the number of requests, e.g. during rush hour, the increases in URT of the static requests are more drastic compared to the increases in URT when the fleet size was relatively large to begin with. For completeness Fig. 12b shows the percentage difference in average URT per passenger for the static requests before static optimization compared to their URT's at the end of the planning horizon. Except for a high degree of dynamism, declines in average URT per passenger can be seen. This is of course because the initial solution of the static requests is not optimized and has consequently relatively high URT's. It shows once again that it is important to do some static optimization when the chances of dynamic requests coming in are scarce.

Figure 13a and Fig. 13b give an insight in the changes in average URT of the dynamic requests. The first plots the percentage difference in average URT per passenger for the dynamic requests in the final solution compared to their URT's after initial insertion. The second plots almost the same, but instead of comparing to their initial insertion, the difference is plotted with their URT's after the 15 dynamic iterations done after their initial insertion. This is the optimization which was caused by their request. After this short optimization the passenger who has submitted the request gets already an estimation of the route duration. Therefore it could be argued the second graph is the most important to analyze. It shows relatively stable URT's for the dynamic requests when the percentage of static requests is higher than 40% and when the fleet size is large enough. Once again the fleet size influences the severity of the URT increases. When operating a fully dynamic on-demand system, one should especially be careful making service promises to the passengers, because in this case for all fleet sizes small and large, the URT of the (dynamic) requests increases with 39 to 48%. This effect is strongly countered once only 20% of the total number of requests is already known in advance. In this case the URT increase is at most 19% for the tightest fleet size, for other fleet sizes 10% on average. Because of these large relative differences, one could argue to base the route duration estimation for the dynamic requests on the route duration of their initial insertion. The relative URT's are given in Fig. 13a. These are more moderate with even declining URT's for large fleet sizes. However, also for the smaller fleet sizes the differences become large for highly dynamic environments.

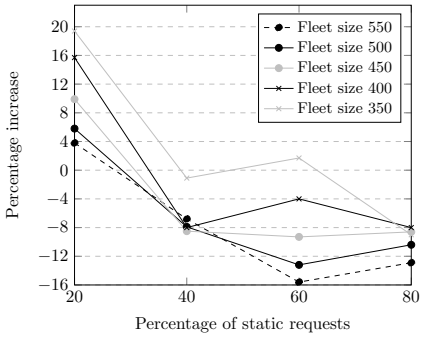
To complete this analysis Fig. 14a and Fig. 14b show the average URT per passenger of the dynamic requests for different degrees of dynamism. The first plots the results when a fleet size of 350 is adopted, the second for a fleet size of 550. By comparing the two graphs the difference in average URT per passenger between a tight and large enough fleet size is directly visible for all degrees of dynamism. In the figures both the URT after initial insertion, as the

URT after the 15 iterations of dynamic optimization and the final URT per passenger are illustrated. The differences in URT discovered in Fig. 13a and Fig. 13b are visible here as well. For the fully dynamic ODBRP, large increases in URT relative to the initial insertion or URT after dynamic optimization can be noticed. What is interesting to see in these figures is the fact that, when the fleet size is large enough (550 buses), the end URT of the dynamic requests is relatively stable. Especially when it is compared to the end URT's of the dynamic requests when the fleet size is tight (350 buses). In the latter a decrease in end URT can be seen for a growing percentage of static requests. The less dynamic the nature of the on-demand bus system, the lower the end URT of the dynamic requests.

To answer the research question posed in the title of this section, one can conclude that dynamic requests are mostly affected by the transition from a static to a dynamic system. Additionally the stability of the URT's is influenced by the fleet size and the level of dynamism. A larger fleet size and lower degree of dynamism cause the URT's of both the static and the dynamic requests to be more steady and consequently easier to estimate as an operating company.

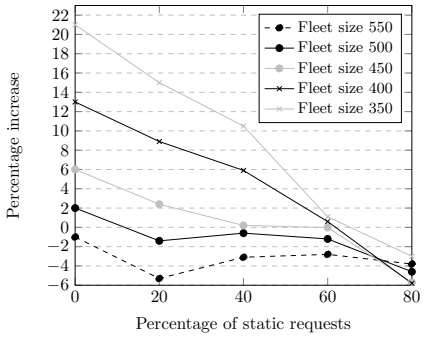


(a) After static optimization

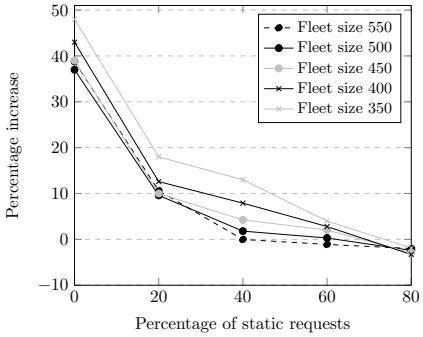


(b) Before static optimization

Figure 12: The percentage difference in average URT per passenger for the static requests

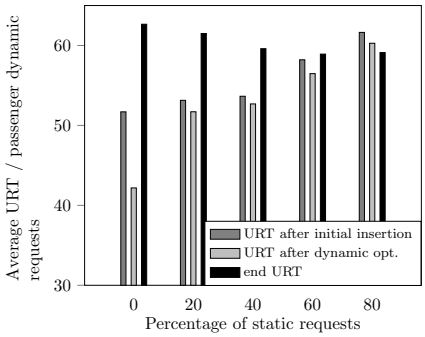


(a) After initial insertion

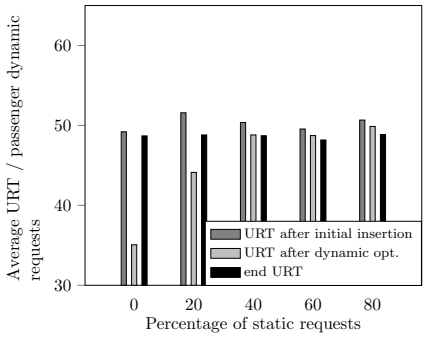


(b) After the 15 dynamic iterations caused by the insertion of the particular dynamic request

Figure 13: The percentage difference in average URT per passenger for the dynamic requests



(a) Fleet size 350

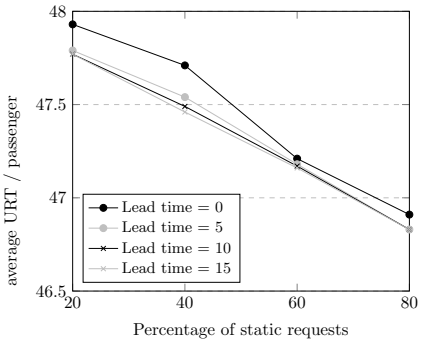


(b) Fleet size 550

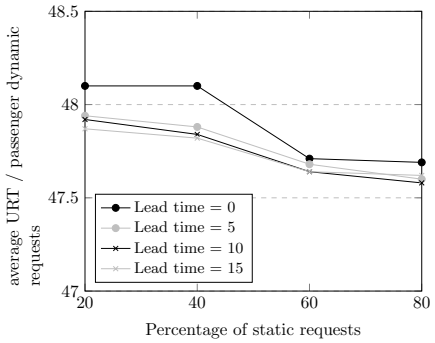
Figure 14: Average URT per passenger for the dynamic requests for different degrees of dynamism and different fleet sizes

4.2.3 Does increasing the lead time of requests help to counter the cost of dynamic requests?

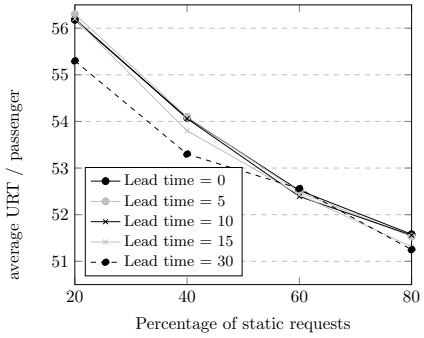
In the previous experiments, we assumed that the dynamic requests came in just-in-time, without lead time. This means passengers indicate they want to leave as soon as possible after sending their request. However, as we already found several benefits of increasing the number of static requests, it is interesting to investigate the influence of the lead time. In this case the request is issued some amount of time before the earliest departure time. Requests with a lead time are not static, but also not highly dynamic and just-in-time. The instances used in the previous sections all had an earliest departure time between time 10 and 70. For this experiment new instances of 2000 requests are generated with earliest departure times between 10 and 130, to better see the effect of the increased lead times. Figure 15a plots once again the percentage of static requests on the x-axis and the average URT per passenger on the y-axis, this time for different lead times. A slight decrease of 0.02% in URT per passenger is visible when the lead time increases from zero to 5 minutes for the higher levels of dynamism. When the situation is more of a static kind (60 to 80% static requests) the difference is barely visible. Likewise, when the lead time is further expanded up to 15 minutes, the difference between a lead time of 5 minutes is hardly noticeable for all levels of dynamism. Therefore we propose not to impose an obligated lead time to future customers of an on-demand bus system. The algorithm is fast enough to deal with just-in-time requests and the benefits for the service quality are minimal. On the other hand, allowing a lead time can be beneficial in terms of the URT of the dynamic requests. In Section 4.2.2 it was clear that the average URT per passenger for the dynamic requests is significantly higher compared to the average URT per passenger of the static requests. An increased lead time helps slightly to balance this difference. Figure 15b shows the effect of an increased lead time on the average URT per passenger of the dynamic requests. Especially for high degrees of dynamism a decline in average URT per passenger is visible.



(a) All requests - fleet size 500



(b) Only dynamic requests - fleet size 500



(c) All requests - fleet size of 350

Figure 15: The effect of the percentage of static requests on the average URT per passenger for all and only for the dynamic requests for different lead times, for instances with an earliest departure time between 10 and 130

The above analysis is done for a problem with a comfortably large fleet size that can easily serve every request (500 buses). When the fleet size is lowered to the point where just every request can be served, this is a fleet size of 350 buses, another trend can be seen (Fig. 15c). As was discovered in previous sections, overall, the URT per passenger increases when lowering the fleet size. But secondly, the minimal effect of an increased lead time visible in Fig. 15a is not noticeable anymore. Even for high degrees of dynamism an increased lead time up to 15 minutes does not cause a decrease in URT. Therefore a lead time of 30 minutes for all requests is tested and as is clear on the figure, this causes a decline in URT for high degrees of dynamism. For smaller fleet sizes a larger lead time needs to be adopted to make a difference. However, overall this difference is small and also with a lower fleet size, it is not worth it to impose a lead time to the dynamic requests.

5 Conclusion and future research

This work introduced the real-time on-demand bus routing problem (ODBRP) for the routing of on-demand buses in a real-time context. The real-time ODBRP differs from the static variant by adding the possibility of real-time requests. This means that not all requests are known beforehand, some of the requests will be issued during the planning horizon. An existing heuristic for the static ODBRP is adapted to incorporate these dynamic requests, while also minimizing the total user ride time (URT). First, the known requests are scheduled in the solution and static optimization is executed to obtain a decent initial solution for the dynamic part. Afterwards, the dynamic requests are added one by one and the heuristic executes a real-time optimization. It is found that the importance of the static optimization is rather insignificant as the initial solution still changes substantially by adding dynamic requests. However, time is not an issue in static optimization. In addition, static optimization is still important when the problem is extremely static, i.e. when most requests are known before the planning horizon. Therefore, we advise

to keep the static optimization in place.

Furthermore the cost of adding the possibility of dynamic requests is investigated, compared to only allowing these requests to be sent before the planning horizon. As time is money, this cost is expressed in a deterioration of the objective function value, the total URT. Overall we conclude that higher degrees of dynamism induce higher costs, both in terms of higher URT's, the stability of the URT's (reliability of the service) and in terms of a larger needed fleet size. It is therefore both from the operator's perspective as from the passengers' perspective beneficial to increase the number of requests known beforehand. For the passenger this causes their average final URT to be lower, compared to passengers who send their request just-in-time. After finding these results, the effect of a small lead time is investigated and it is found that imposing a lead time of 5 to 15 minutes does not cause a generous decline in URT and is therefore not recommended.

Both the absolute levels of URT and their stability worsen when the fleet size is small compared to the total number of requests. Therefore we especially recommend to encourage future passengers to send their requests in advance during rush hours. Passengers can be persuaded to do this by both the promise of a lower URT and by offering a lower price.

Future research is needed to further investigate the willingness to pay of passengers and the tactics of differentiated pricing. For example, lower prices can be asked for sending requests in advance or when the fleet size is relatively large compared to the total number of requests, during off-peak hours. Moreover, there is opportunity for a more sophisticated heuristic that focuses on a quick and efficient insertion of the dynamic requests to be able to handle even larger instances with a larger fleet size of buses. Finally, to further increase the realism of the problem, real-time traffic information and instances based on real cities can be used.

Acknowledgements

This research was supported by a grant of the Flemish Fund for Scientific Research (FWO Strategic Basic Research Fellowship grant).

References

Agatz, N., Erera, A., Savelsbergh, M., and Wang, X. (2012). Optimization for dynamic ride-sharing: A review. *European Journal of Operational Research*, 223(2):295–303.

Alonso-Mora, J., Samaranayake, S., Wallar, A., Frazzoli, E., and Rus, D. (2017). On-demand high-capacity ride-sharing via dynamic trip-vehicle assignment. *Proceedings of the National Academy of Sciences*, 114(3):462–467.

Archetti, C., Speranza, M. G., and Weyland, D. (2018). On-demand public transportation. *Int. Trans. Oper. Res.*

Bischoff, J., Führer, K., and Maciejewski, M. (2018). Impact assessment of autonomous drt systems. *Transportation Research Procedia*, pages 1–10.

Brevet, D., Duhamel, C., Iori, M., and Lacomme, P. (2019). A dial-a-ride problem using private vehicles and alternative nodes. *Journal on Vehicle Routing Algorithms*, 2(1):89–107.

Brownstone, D., Ghosh, A., Golob, T. F., Kazimi, C., and Van Amelsfort, D. (2003). Drivers' willingness-to-pay to reduce travel time: evidence from the san diego i-15 congestion pricing project. *Transportation Research Part A: Policy and Practice*, 37(4):373–387.

Bruni, M., Guerriero, F., and Beraldi, P. (2014). Designing robust routes for demand-responsive transport systems. *Transportation research part E: logistics and transportation review*, 70:1–16.

Cordeau, J.-F. and Laporte, G. (2007). The dial-a-ride problem: models and algorithms. *Annals of operations research*, 153(1):29–46.

Czioska, P., Kutadinata, R., Trifunović, A., Winter, S., Sester, M., and Friedrich, B. (2019). Real-world meeting points for shared demand-responsive transportation systems. *Public Transport*, 11(2):341–377.

Gökay, S., Heuvels, A., and Krempels, K. (2019). On-demand ride-sharing services with meeting points. In *5th International Conference on Vehicle Technology and Intelligent Transport Systems (VEHITS 2019)*, pages 117–125.

Häll, C. H., Andersson, H., Lundgren, J. T., and Värbrand, P. (2009). The integrated dial-a-ride problem. *Public Transport*, 1(1):39–54.

Ho, S. C., Szeto, W., Kuo, Y.-H., Leung, J. M., Petering, M., and Tou, T. W. (2018). A survey of dial-a-ride problems: Literature review and recent developments. *Transportation Research Part B: Methodological*, 111:395–421.

Jokinen, J.-P., Sihvola, T., Hyytiä, E., and Sulonen, R. (2011). Why urban mass demand responsive transport? In *2011 IEEE Forum on Integrated and Sustainable Transportation Systems*, pages 317–322. IEEE.

Koh, K., Ng, C., Pan, D., and Mak, K. S. (2018). Dynamic bus routing: A study on the viability of on-demand high-capacity ridesharing as an alternative to fixed-route buses in singapore. In *2018 21st International Conference on Intelligent Transportation Systems (ITSC)*, pages 34–40. IEEE.

Li, Z., Hensher, D. A., and Rose, J. M. (2010). Willingness to pay for travel time reliability in passenger transport: A review and some new empirical evidence. *Transportation research part E: logistics and transportation review*, 46(3):384–403.

Melis, L. and Sörensen, K. (2020). The on-demand bus routing problem: A large neighborhood search heuristic for a dial-a-ride problem with bus station assignment. Working papers, University of Antwerp, Faculty of Business and Economics.

Molenbruch, Y., Braekers, K., and Caris, A. (2017). Typology and literature review for dial-a-ride problems. *Annals of Operations Research*, 259(1-2):295–325.

Navidi, Z., Ronald, N., and Winter, S. (2018). Comparison between ad-hoc demand responsive and conventional transit: a simulation study. *Public Transport*, 10(1):147–167.

Pavone, M. (2015). Autonomous mobility-on-demand systems for future urban mobility. In *Autonomes Fahren*, pages 399–416. Springer.

Posada, M., Andersson, H., and Häll, C. H. (2017). The integrated dial-a-ride problem with timetabled fixed route service. *Public Transport*, 9(1-2):217–241.

Stiglic, M., Agatz, N., Savelsbergh, M., and Gradisar, M. (2018). Enhancing urban mobility: Integrating ride-sharing and public transit. *Computers & Operations Research*, 90:12–21.

Tsao, M., Milojevic, D., Ruch, C., Salazar, M., Frazzoli, E., and Pavone, M. (2019). Model predictive control of ride-sharing autonomous mobility-on-demand systems. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 6665–6671. IEEE.

Uber (2016). Two years of uberPOOL. Online; accessed 18 September 2018.

Uber (2019). How it works. Online; accessed 12 February 2019.

Vallée, S., Oulamara, A., and Cherif-Khettaf, W. R. (2017). Maximizing the number of served requests in an online shared transport system by solving a dynamic darp. In *International Conference on Computational Logistics*, pages 64–78. Springer.

van Engelen, M., Cats, O., Post, H., and Aardal, K. (2018). Enhancing flexible transport services with demand-anticipatory insertion heuristics. *Transportation Research Part E: Logistics and Transportation Review*, 110:110–121.

Winter, K., Cats, O., Correia, G., and van Arem, B. (2018). Performance analysis and fleet requirements of automated demand-responsive transport systems as an urban public transport service. *International Journal of Transportation Science and Technology*, 7(2):151–167.

Yan, X., Zhao, X., Han, Y., Van Hentenryck, P., and Dillahunt, T. (2019). Mobility-on-demand versus fixed-route transit systems: an evaluation of traveler preferences in low-income communities. *arXiv preprint arXiv:1901.07607*.

Appendix

A Pseudo-codes algorithm

Algorithm 1: LNS for the real-time ODBRP [a]

```
1  Function MaxP( iterations):
2      while R < |P| and iterations < static iterations do
3          Destriy and Repair (maxP);
4          iterations++;
5      if R = |P| then
6          while iterations < static iterations do
7              Local search 20 iterations;
8              iterations += 20;
9  Function LNS static():
10     Build initial solution with greedy constructive heuristic;
11     iterations = 0;
12     count = 0;
13     if P ≠ |R| then
14         maxP(iterations);
15     while iterations < static iterations do
16         Destroy and Repair (minURT);
17         iterations++;
18         if P = |R| then
19             Local search;
20             count = 0 ;
21         else
22             if count = 5 then
23                 count = 0;
24                 maxP(iterations);
25             else
26                 count++;
27 Function Main():
28     LNS static();
29     for every dynamic request r do
30         Current time = earliest departure[r] - lead time - stop time ;
31         |R|++ ;
32         LP = Define Locking Point(Current time) ;
33         Insert r in solution after LP with greedy constructive heuristic;
34         iterations = 0 ;
35         if P = |R| then
36             while iterations < dynamic iterations do
37                 Destroy and repair (minURT) after LP ;
38                 iterations++ ;
39                 if P = |R| then
40                     Local search ;
41         else
42             while P < |R| and until stop criterion do
43                 Destroy and repair (maxP) after LP ;
44             if P = |R| then
45                 while iterations < dynamic iterations do
46                     Destroy and repair minURT after LP ;
47                     iterations++ ;
48                     if P = |R| then
49                         Local search ;
50             else
51                 Go back to solution before trying to insert r;
```

[a] P the number of served passengers and $|R|$ the number of passengers known

Algorithm 2: Algorithm for defining locking point

```
1  Function Define Locking Point(current time):
2      Locking point = 0;
3      for every stop s in route b do
4          if arrival time at stop s < current time then
5              Locking point = s ;
6          else
7              Break out of loop ;
8      if Locking point is not last scheduled stop in route then
9          Locking point ++ ;
10     if Locking point is not last scheduled stop in route - 1 then
11         for Every stop s between Locking point and last scheduled stop in route -1 do
12             if Someone gets on bus at stop s then
13                 bool breaknow = true;
14                 if departure time at stop s - stop time - walking time < current time then
15                     breaknow = false;
16                     Locking point ++;
17                 if breaknow then
18                     Break out of loop;
19     return Locking point;
```

B Example of infeasible reconstruction after destroy and repair operator

In Fig. B1 four requests are scheduled in a route of which the first stop is fixed. LNS is performed, clearing stop 2 to 4. However P6, P7 and P8 get on the bus at the first stop and need to be re-added in the route. P9 can be rescheduled elsewhere. They are inserted in the same order as they were scheduled to get off the bus in the original bus route and in a greedy manner. However this time P6 is scheduled to get off at station 3 instead of 2. This causes the algorithm to choose a different station for P7 as well, as station 7 is closer to station 3 instead of station 6. P8 can only get off at station 4. However the distance between station 6 and 4 is significantly smaller than the distance between station 7 and 4. This results in an infeasible solution, because P8 arrives at time 60 which is after the latest arrival time of 59.

Requests	P	e_p^u	l_p^o	Departure stations	Arrival stations
	6	0	25	1	2 or 3
	7	0	28	1	6 or 7
	8	0	**59**	1	4
	9	5	48	2 or 3	6 or 7

Bus stop	1	2	3	4
Bus station	1	2	6	4
Arrival time	0	6	21	57
Departure time	1	7	22	58
P on	6,7,8	9		
P off		6	7,9	8

Bus stop	1	2	3	4
Bus station	1	3	7	4
Arrival time	0	6	19	**60**
Departure time	1	7	20	61
P on	6,7,8			
P off		6	7	8

Figure B1: Additional example of an unfeasible reconstruction of arrival stops after LNS destroy operator